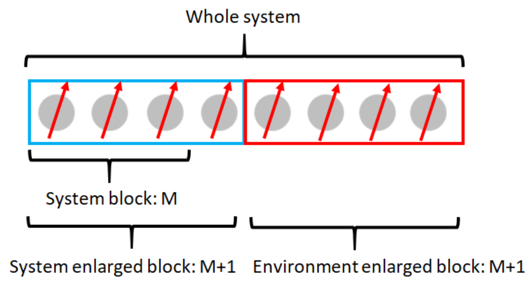
Part 1: Basic algorithm

The basic algorithm goes like this:

1, build the representation of a system composed of N=2M+2 sites using the infinite DMRG algorithm. Here M is the number of sites we are interested in.

$$\widehat{H}_N = \widehat{H}_{M+1} + \widehat{H}_{\text{int}} + \widehat{H}_{1+M}$$

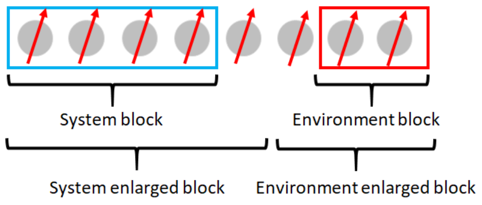
During this process, store all the left and right blocks, with their corresponding operators and basis transformations.



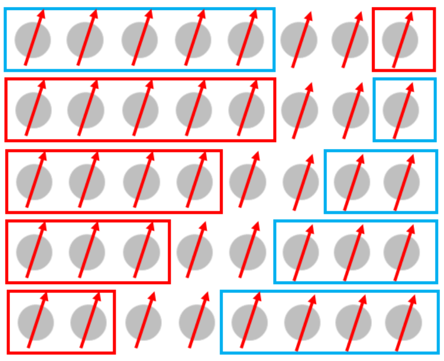
2. start a new DMRG calculation.
Here the system enlarged block of last step is now the system block. Since when building the new system, Hamiltonian to keep the system size fixed to N, we have:

$$\widehat{H}_N = \widehat{H}_{M+1+1} + \widehat{H}_{\text{int}} + \widehat{H}_M$$

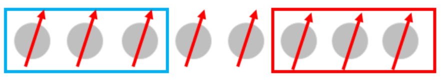
Notice that the left and right enlarged block now are different and represent a different number of sites. Obtain the truncated representation of the Hamiltonian $\widehat{\widehat{H}}_{M+1+1}$.



3. keep on iterating, increasing the size of the left block and decreasing that of the right block, keeping N constant. When the boundary is reached, reverse the process and keep iterating inverting the role of the left and right block.



Continue .....

Until 

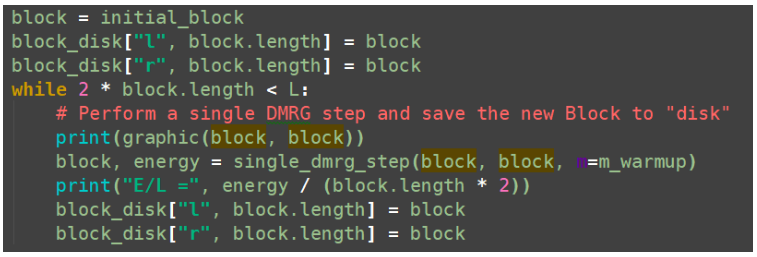
Part 2: Object orientated code

Here we base on the code of infinite DMRG.

The total system now has fixed number of site, and the system block as well the environment block varies. Here we define a dictionary:

block_disk={}

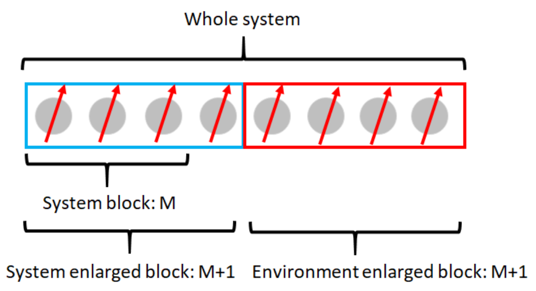
to store both the block information:



```
block = initial_block
block_disk["l", block.length] = block
block_disk["r", block.length] = block
while 2 * block.length < L:
    # Perform a single DMRG step and save the new Block to "disk"
    print(graphic(block, block))
    block, energy = single_dmrg_step(block, block, m=m_warmup)
    print("E/L =", energy / (block.length * 2))
    block_disk["l", block.length] = block
    block_disk["r", block.length] = block
```

For example, suppose we have L=8, we have block.length=3 for the last recursion, and the output block will have block.length=4.

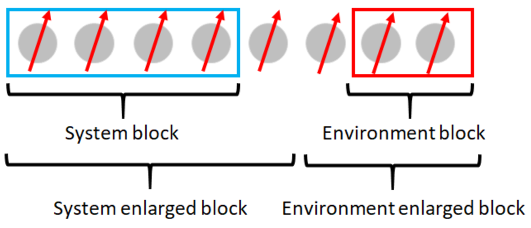
Therefore, we have block for both system enlarged block and environment enlarged block:



Up to here, the chain with desired size is built up.

Now start sweep.

At first iteration, we have



i.e.

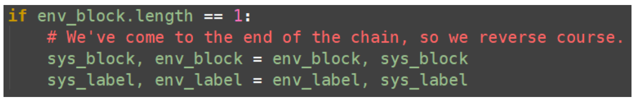
sys_block = block   # size = 4

env_block = block_disk[env_label, L-sys_block.length-2] # size = 2

sys_block, energy = single_dmrg_step(sys_block, env_block, m=m)

block_disk [sys_label, sys_block.length] = sys_block

And when we come to the end of chain, system block and environment block should exchange place:



```
if env_block.length == 1:
    # We've come to the end of the chain, so we reverse course.
    sys_block, env_block = env_block, sys_block
    sys_label, env_label = env_label, sys_label
```

And to check whether one full sweep is done, we need to check the label of sys_block and the length of sys_block:

```
if sys_label == "l" and 2 * sys_block.length == L:
    break  # escape from the "while True" loop
```

Therefore, we have the main code for finite_system_algorithm:

```
def finite_system_algorithm(L, m_warmup, m_sweep_list):
    assert L % 2 == 0  # require that L is an even number

    # To keep things simple, this dictionary is not actually saved to
disk, but we use it to represent persistent storage.
    block_disk = {}  # "disk" storage for Block objects

    # Use the infinite system algorithm to build up to desired size.
Each time we construct a block, we save it for future reference as
both a left ("l") and right ("r") block, as the infinite system
algorithm assumes the environment is a mirror image of the system.
    block = initial_block
    block_disk["l", block.length] = block
    block_disk["r", block.length] = block
    while 2 * block.length < L:
        # Perform a single DMRG step and save the new Block to "disk"
        print(graphic(block, block))
        block, energy = single_dmrg_step(block, block, m=m_warmup)
        print("E/L =", energy / (block.length * 2))
        block_disk["l", block.length] = block
        block_disk["r", block.length] = block

    # Now that the system is built up to its full size, we perform sweeps
using the finite system algorithm.  At first the left block will act
as the system, growing at the expense of the right block (the
environment), but once we come to the end of the chain these roles
will be reversed.
    sys_label, env_label = "l", "r"
    sys_block = block; del block  # rename the variable
    for m in m_sweep_list:
        while True:
            # Load the appropriate environment block from "disk"
            env_block = block_disk[env_label, L - sys_block.length -
2]
            if env_block.length == 1:
                # We've come to the end of the chain, so we reverse course.
                sys_block, env_block = env_block, sys_block
                sys_label, env_label = env_label, sys_label
            # Perform a single DMRG step.
            print(graphic(sys_block, env_block, sys_label))
            sys_block, energy = single_dmrg_step(sys_block, env_block, 
m=m)

            print("E/L =", energy / L)

            # Save the block from this step to disk.
            block_disk[sys_label, sys_block.length] = sys_block

            # Check whether we just completed a full sweep.
            if sys_label == "l" and 2 * sys_block.length == L:
                break  # escape from the "while True" loop
```

Call the function:

```
if __name__ == "__main__":
    np.set_printoptions(precision=10, suppress=True, 
threshold=10000, linewidth=300)

    #infinite_system_algorithm(L=100, m=20)
    finite_system_algorithm(L=20, m_warmup=10, m_sweep_list=[10])
```

Visualization